Assignment 19 Solutions

1. Merge k Sorted Lists

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: lists = [[1,4,5],[1,3,4],[2,6]]

Output: [1,1,2,3,4,4,5,6]

Explanation: The linked-lists are:

[1->4->5,

1->3->4,

2->6]

merging them into one sorted list:

1->1->2->3->4->4->5->6

Example 2:

Input: lists = []

Output: []

Example 3:

Input: lists = [[]]

Output: []

****Constraints:**

- `k == lists.length`
- `0 <= k <= 10000`
- `0 <= lists[i].length <= 500`
- `-10000 <= lists[i][j] <= 10000`
- `lists[i]` is sorted in **ascending order**.
- The sum of `lists[i].length` will not exceed `10000`.

In [76]:
```python
# Example 1
import heapq

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

    def __lt__(self, other):
        return self.val < other.val

def mergeKLists(lists):
    heap = []

    for head in lists:
        if head:
            heapq.heappush(heap, head)

    # Initialize a dummy node
    dummy = ListNode()
    curr = dummy

    while heap:
        node = heapq.heappop(heap)

        curr.next = node
        curr = curr.next

        if node.next:
```

```
            heapq.heappush(heap, node.next)

    return dummy.next
```

In [77]:
```
lists = [[ListNode(val) for val in sublist] for sublist in [[1,4,5],[1,3,4],[2,6]]]
for i in range(len(lists)):
    for j in range(len(lists[i])-1):
        lists[i][j].next = lists[i][j+1]

merged_list = mergeKLists([sublist[0] for sublist in lists if sublist])

output = []
while merged_list:
    output.append(merged_list.val)
    merged_list = merged_list.next

print(output)
```

```
[1, 1, 2, 3, 4, 4, 5, 6]
```

In [78]:
```
lists = []
output = mergeKLists(lists)
print(output)
```

```
None
```

In [79]:
```
lists = [[]]
output = mergeKLists(lists)
print(output)
```

```
None
```

2. Count of Smaller Numbers After Self

Given an integer array `nums`, return *an integer array* `counts` *where* `counts[i]` *is the number of smaller elements to the right of* `nums[i]`.

Example 1:

Input: nums = [5,2,6,1]

Output: [2,1,1,0]

Explanation:

To the right of 5 there are2 smaller elements (2 and 1).

To the right of 2 there is only1 smaller element (1).

To the right of 6 there is1 smaller element (1).

To the right of 1 there is0 smaller element.

Example 2:

Input: nums = [-1]

Output: [0]

Example 3:

Input: nums = [-1,-1]

Output: [0,0]

Constraints:

- `1 <= nums.length <= 100000`
- `-10000 <= nums[i] <= 10000`

In [80]:
```
def countSmaller(nums):
    counts = [0] * len(nums)
    indices = list(range(len(nums)))

    def mergeSort(start, end):
        if start < end:
            mid = (start + end) // 2
            mergeSort(start, mid)
            mergeSort(mid + 1, end)
            merge(start, mid, end)
```

```
    def merge(start, mid, end):
        left = start
        right = mid + 1
        merged = []

        while left <= mid and right <= end:
            if nums[indices[left]] <= nums[indices[right]]:
                merged.append(indices[left])
                counts[indices[left]] += right - mid - 1
                left += 1
            else:
                merged.append(indices[right])
                right += 1

        while left <= mid:
            merged.append(indices[left])
            counts[indices[left]] += end - mid
            left += 1

        while right <= end:
            merged.append(indices[right])
            right += 1

        indices[start:end+1] = merged

    mergeSort(0, len(nums) - 1)

    return counts
```

In [81]:
```
nums = [5, 2, 6, 1]
result = countSmaller(nums)
print(result)
```

```
[2, 1, 1, 0]
```

In [82]:
```
nums = [-1]
result = countSmaller(nums)
print(result)
```

```
[0]
```

In [83]:
```
nums = [-1,-1]
result = countSmaller(nums)
print(result)
```

```
[0, 0]
```

3. Sort an Array

Given an array of integers `nums`, sort the array in ascending order and return it.

You must solve the problem **without using any built-in** functions in `O(nlog(n))` time complexity and with the smallest space complexity possible.

Example 1:

Input: nums = [5,2,3,1]

Output: [1,2,3,5]

Explanation: After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).

Example 2:

Input: nums = [5,1,1,2,0,0]

Output: [0,0,1,1,2,5]

Explanation: Note that the values of nums are not necessairly unique.

Constraints:

- `1 <= nums.length <= 5 * 10000`
- `-5 * 104 <= nums[i] <= 5 * 10000`

In [84]:
```
def sortArray(nums):
    def mergeSort(arr):
        if len(arr) <= 1:
            return arr

        mid = len(arr) // 2
```

```
        left = mergeSort(arr[:mid])
        right = mergeSort(arr[mid:])

        merged = []
        i, j = 0, 0

        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                merged.append(left[i])
                i += 1
            else:
                merged.append(right[j])
                j += 1

        merged.extend(left[i:])
        merged.extend(right[j:])

        return merged

    return mergeSort(nums)
```

In [85]:
```
# Example 1
nums = [5, 2, 3, 1]
output = sortArray(nums)
print(output)

# Example 2
nums = [5, 1, 1, 2, 0, 0]
output = sortArray(nums)
print(output)
```

```
[1, 2, 3, 5]
[0, 0, 1, 1, 2, 5]
```

4. Move all zeroes to end of array

Given an array of random numbers, Push all the zero's of a given array to the end of the array. For example, if the given arrays is {1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0}, it should be changed to {1, 9, 8, 4, 2, 7, 6, 0, 0, 0, 0}. The order of all other elements should be same. Expected time complexity is O(n) and extra space is O(1).

Example:

Input : arr[] = {1, 2, 0, 4, 3, 0, 5, 0};

Output : arr[] = {1, 2, 4, 3, 5, 0, 0, 0};

Input : arr[] = {1, 2, 0, 0, 0, 3, 6};

Output : arr[] = {1, 2, 3, 6, 0, 0, 0};

In [86]:
```
def moveZeroes(nums):
    left = 0
    n = len(nums)

    for right in range(n):
        if nums[right] != 0:
            nums[left], nums[right] = nums[right], nums[left]
            left += 1

    while left < n:
        nums[left] = 0
        left += 1

    return nums
```

In [87]:
```
# Example 1
nums = [1, 2, 0, 4, 3, 0, 5, 0]
output = moveZeroes(nums)
print(output)

# Example 2
nums = [1, 2, 0, 0, 0, 3, 6]
output = moveZeroes(nums)
print(output)
```

```
[1, 2, 4, 3, 5, 0, 0, 0]
[1, 2, 3, 6, 0, 0, 0]
```

5. Rearrange array in alternating positive & negative items with O(1) extra space

Given an **array of positive** and **negative numbers**, arrange them in an **alternate** fashion such that every positive number is followed by a negative and vice-versa maintaining the **order of appearance**. The number of positive and negative numbers need not be equal. If

there are more positive numbers they appear at the end of the array. If there are more negative numbers, they too appear at the end of the array.

Examples:

Input: arr[] = {1, 2, 3, -4, -1, 4}

Output: arr[] = {-4, 1, -1, 2, 3, 4}

Input: arr[] = {-5, -2, 5, 2, 4, 7, 1, 8, 0, -8}

Output: arr[] = {-5, 5, -2, 2, -8, 4, 7, 1, 8, 0}

In [88]:
```
def rightRotate(arr, n, outOfPlace, cur):
    temp = arr[cur]
    for i in range(cur, outOfPlace, -1):
        arr[i] = arr[i - 1]
    arr[outOfPlace] = temp
    return arr


def rearrange(arr, n):
    outOfPlace = -1
    for index in range(n):
        if(outOfPlace >= 0):
            if((arr[index] >= 0 and arr[outOfPlace] < 0) or
            (arr[index] < 0 and arr[outOfPlace] >= 0)):
                arr = rightRotate(arr, n, outOfPlace, index)
                if(index-outOfPlace > 2):
                    outOfPlace += 2
                else:
                    outOfPlace = - 1

        if(outOfPlace == -1):

            if((arr[index] >= 0 and index % 2 == 0) or
            (arr[index] < 0 and index % 2 == 1)):
                outOfPlace = index
    return arr
```

In [89]:
```
arr = [1, 2, 3, -4, -1, 4]
print(rearrange(arr, len(arr)))
```

```
[-4, 1, -1, 2, 3, 4]
```

In [90]:
```
arr = [-5, -2, 5, 2, 4, 7, 1, 8, 0, -8]
print(rearrange(arr, len(arr)))
```

```
[-5, 5, -2, 2, -8, 4, 7, 1, 8, 0]
```

6. Merge two sorted arrays

Given two sorted arrays, the task is to merge them in a sorted manner.

Examples:

Input: arr1[] = { 1, 3, 4, 5}, arr2[] = {2, 4, 6, 8}

Output: arr3[] = {1, 2, 3, 4, 4, 5, 6, 8}

Input: arr1[] = { 5, 8, 9}, arr2[] = {4, 7, 8}

Output: arr3[] = {4, 5, 7, 8, 8, 9}

In [91]:
```
def mergeArrays(arr1, arr2):
    merged = []
    i = 0
    j = 0

    while i < len(arr1) and j < len(arr2):
        if arr1[i] <= arr2[j]:
            merged.append(arr1[i])
            i += 1
        else:
            merged.append(arr2[j])
            j += 1

    while i < len(arr1):
        merged.append(arr1[i])
        i += 1
```

```
    while j < len(arr2):
        merged.append(arr2[j])
        j += 1

    return merged
```

In [92]:
```
# Example 1
arr1 = [1, 3, 4, 5]
arr2 = [2, 4, 6, 8]
output = mergeArrays(arr1, arr2)
print(output)

# Example 2
arr1 = [5, 8, 9]
arr2 = [4, 7, 8]
output = mergeArrays(arr1, arr2)
print(output)
```

```
[1, 2, 3, 4, 4, 5, 6, 8]
[4, 5, 7, 8, 8, 9]
```

7. Intersection of Two Arrays

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must be **unique** and you may return the result in **any order**.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [9,4]

Explanation: [4,9] is also accepted.

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 1000`

In [93]:
```
def intersection(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)
    return list(set1.intersection(set2))
```

In [94]:
```
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
result1 = intersection(nums1, nums2)
print(result1)
```

```
[2]
```

In [95]:
```
nums3 = [4, 9, 5]
nums4 = [9, 4, 9, 8, 4]
result2 = intersection(nums3, nums4)
print(result2)
```

```
[9, 4]
```

8. Intersection of Two Arrays II

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2,2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [4,9]

Explanation: [9,4] is also accepted.

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 1000`

In [96]:
```python
from collections import defaultdict

def intersect(nums1, nums2):
    freqMap = defaultdict(int)
    intersection = []

    for num in nums1:
        freqMap[num] += 1

    for num in nums2:
        if freqMap[num] > 0:
            intersection.append(num)
            freqMap[num] -= 1

    return intersection
```

In [97]:
```python
# Example 1
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
output = intersect(nums1, nums2)
print(output)

# Example 2
nums1 = [4, 9, 5]
nums2 = [9, 4, 9, 8, 4]
output = intersect(nums1, nums2)
print(output)
```

```
[2, 2]
[9, 4]
```

In [ ]: